Name: Hanson Ma   RCS ID: mah11 @rpi.edu

CSCI 4210 Operating Systems — Exam 2 — April 12, 2023

Overview

- This is a 110-minute exam (6:00-7:50PM); if you have extra-time accommodations, then you have either 165 minutes (50%) or 220 minutes (100%) and should write your start and end times at the top of this page
- No calculators, cellphones, etc.; please be sure to turn off electronic devices before the exam
- Questions will **not** be answered except when there is a mistake or ambiguity in the statement of a question; please do your best to interpret and answer each question
- There are 12 questions on this exam, including two extra credit questions; this exam will be graded out of 60 possible points; with the extra credit questions, it is possible to get a 62
- Do **not** detach any pages; further, anything written on the back of any pages of this exam will **not** be graded
- You are allowed to use two double-sided or four single-sided 8.5"x11" cribsheets
- **All work on this exam must be your own; do not copy or communicate with anyone else about this exam until the exam is graded**
- Once we have graded your exam, solutions will be posted; the grade inquiry window (in Submitty) for this exam will then be one week

Assumptions

1. Assume that all given code compiles without warnings or errors (unless otherwise noted); assume all header files are included
2. Assume that all library function and system calls return successfully (unless otherwise noted)
3. Assume that all code runs on a 64-bit Ubuntu 20.04.5 LTS instance (unless otherwise noted)
4. Assume that threads initially run with a thread ID of 32; child thread IDs are then numbered sequentially as they are created starting at 64 (i.e., 64, 65, 66, etc.)

Academic Integrity Confirmation

Please sign below to confirm that you will not copy and you will not cheat on this exam, which also means that you will not communicate with anyone under any circumstances about this exam until it is graded:

Signature: H.M

Any copying or collaborating with others will result in a grade of zero on this exam.

3510

1. **(3 POINTS)** When you call pthread_detach(), what happens? Circle the **best** answer.

   (a) The thread detaches by terminating and returning NULL

   (b) The thread creates a shared memory segment that is joined across all threads

   [✓ (c)] The thread is set up to not be joined by its parent thread

   (d) A child thread is acknowledged as terminated

   (e) Memory is copied from one thread to another, i.e., they "join" resources

   (f) A new thread is created in a detached form within the running process

2100
2310

2. **(3 POINTS)** How many total bytes are allocated on the runtime heap after the free() call completes in the code snippet shown below? Circle the **best** answer.

```
void * q2( void * a )
{
  char * two = calloc( *((int *)a) + 2000, sizeof( char ) );   2000  + '\0'
  for ( int b = 1 ; b <= sizeof( two ) ; b++ )
  {
    sprintf( two, "abcdefghijklmnopqrstuvwxyz" );
    two = realloc( two, 90 + 135 * b );
  }
  return NULL;
}

int main()
{
  int * sage = calloc( 3101, sizeof( int ) );   3101 · 4
  pthread_t t1, t2, t3;
  pthread_create( &t1, NULL, q2, sage );
  pthread_create( &t2, NULL, q2, sage );
  pthread_create( &t3, NULL, q2, sage );
  pthread_join( t3, NULL );
  pthread_join( t2, NULL );
  pthread_join( t1, NULL );
  free( sage );
  return EXIT_SUCCESS;
}
```

0 (above `*((int *)a)`), 8 (below `sizeof( two )`)

8   90 + 135 · 8
= 770 bytes

x00
+ 240
+ 440
= 680

770 × 3
= 2100
+ 210

(a) 3101

(b) 3303

[✓ (c)] 3510

(d) 3704

(e) 4101

(f) 5101

4

15 = 0b00000111 → class c

3. **(3 POINTS)** What class of network is `ucla.edu`, which has an IP address of `15.197.181.170`? Circle the **best** answer.

   (a) Class A
   (b) Class B
   [✓ (c)] Class C
   (d) Class D (Multicast)
   (e) Class 1 (since UCLA was one of the first four nodes of the Internet)
   (f) Not enough information is provided about domain `ucla.edu`

7

4. **(3 POINTS)** In the code below, assume the `4.txt` file does not initially exist. After this code runs, what is the **exact** contents of the `4.txt` file? Circle the **best** answer.

```
int main()
{
  short * q4 = calloc( 2, sizeof( short ) );      4 bytes
  *q4 = htons( 511 );
  *(q4 + 1) = htons( 255 );
  int fd = open( "4.txt", O_WRONLY | O_CREAT, 0644 );
  write( fd, q4, sizeof( short ) );
  close( fd );              2         16 b
  return EXIT_SUCCESS;
}
```

   (a) 0xff01ff00
   (b) 0x01ff00ff
   (c) 0xff00
   (d) 0xff01
   [✓ (e)] 0x00ff
   ~~(f)~~ 0x01ff

5. **(3 POINTS)** When you call `shmget()`, what happens? Circle the **best** answer.

   (a) The process detaches from the already attached shared memory segment
   (b) A shared memory segment is created and attached to the process
   (c) Given a shared memory ID, the size of the shared memory segment is returned
   [✓ (d)] Given a shared memory key, the shared memory ID is returned
   (e) Given a shared memory ID, the shared memory key is returned
   (f) A shared memory segment with a specified size (in bytes) is attached to the process

66 + 66 e 2   120   132 ea $\bigcirc$198

6. **(10 POINTS)** Assume that initially there is no shared memory segment with key 6666. Also assume that local file alpha.txt contains "ABCDEFGHIJKLM>nopqrstuvwxyz\n"; review the given C code, then answer the questions below.

assume this line always works..

```
int main()
{
  key_t key = 6666;          bytes
  int shmid = shmget( key, 66, IPC_CREAT | 0666 ); ,  <-
  char * six = shmat( shmid, NULL, 0 );   66 '\0'
  pid_t pid = fork();
  if ( pid > 0 ) waitpid( pid, NULL, 0 );     part works fine
  int fd = open( "alpha.txt", O_RDONLY );
  char * buffer = calloc( 66, sizeof( char ) );
  read( fd, buffer, 6 );
  read( fd, buffer, 6 );  GHIJKL -> Buf[0]
#~~ifdef PART_A~~
  ~~if ( pid == 0 ) strcpy( six, buffer + 2 );~~  ~~IJKL~~
#~~endif~~
  read( fd, buffer, 6 );  M>nopq   <- child has buf
  close( fd );
  fprintf( stderr, "<%s>", six );  "IJKL" -> EJKL ign
#ifdef PART_B
  if ( pid == 0 ) strcat( six, buffer + 2 );   IJKL nopqnopq
#endif
  shmdt( six );
  return EXIT_SUCCESS;
}
```

(a) Given only PART_A is set at compile time, what is the exact terminal output of the code?

<IJKL> <IJKL>

(b) Assume that part (a) is done, i.e., one successful execution is complete. If the code is recompiled with only PART_B set, and you run this new executable twice, what is the exact terminal output of the last program execution? Note that strcat() concatenates (i.e., appends) the string pointed to by the second argument to the end of the string pointed to by the first argument.

~~<IJKL~~   <IJKL nopq> <IJKL nopq nopq>

6

4+

7. **(14 POINTS)** Consider the C code below.

```
void * seven( void * arg )
{                                          64...
  printf( "%lu#%d\n", pthread_self(), *(int *)arg );
  fprintf( stderr, "-%lu\n", pthread_self() );
  return NULL;
}

int main()
{
  close( Q7Q7 );
  int * rc = calloc( 1, sizeof( int ) );  0
  printf( "%luX\n", pthread_self() ); 32
  fprintf( stderr, "%lu<<\n", pthread_self() );
  int fd = open( "7.txt", O_WRONLY | O_CREAT | O_TRUNC, 0660 );
  printf( "%luV\n", pthread_self() );
  fprintf( stderr, "7%lu.%d\n", pthread_self(), fd );
  pthread_t tid1, tid2;
  *rc = pthread_create( &tid1, NULL, seven, rc ); 0
  *rc = pthread_create( &tid2, NULL, seven, &fd ); 1
  pthread_join( tid2, NULL );
  pthread_join( tid1, NULL );
  fflush( NULL ); close( fd );
  return EXIT_SUCCESS;
}
```

(a) What is the **exact** terminal output of this program if Q7Q7 is defined as 1? Show all possible outputs.

no stdout →

```
32<<
732.1
-64
-65
```

interleaving ⟷ {-64, -65}

(b) From part (a), what is the **exact** contents of the 7.txt file? Show all possible outputs; if the file is empty, write <empty>.

```
32V
64#0
65#1
```

} interleaving

(c) If Q7Q7 is defined as 2 instead of 1, what is the **exact** contents of the 7.txt file? Show all possible outputs; if the file is empty, write <empty>.

```
32<<
732.2
-64
-65
```

} interleaving

0

8. **(12 POINTS)** Consider the C code below.

```
void * eight( void * arg )
{
  int * s = (int *)arg;
  printf( "%lu eight %d\n", pthread_self(), *s );
  return NULL;
}

int main()
{
  pthread_t tid1, tid2;
  int rc = 0, q8 = 88, x = 8;                          ↗ 8
  rc = pthread_create( &tid1, NULL, eight, &x );      ↗ 8
hang → rc = pthread_create( &tid2, NULL, eight, &x );
  x *= 8 + rc; 64
  rc = pthread_join( tid1, NULL ); 0
  x += 8; 72
  printf( "-%d main %d\n", rc, x );
  if ( ~~q8 == 88~~ ) rc = pthread_join( tid2, NULL );
  return EXIT_SUCCESS;
}
```

64
↕

(a) What is the exact terminal output of the given code? Show all possible outputs.

```
64 eight 8     64/ | 64 eight 8   | 64 eight 8   | 64 eight 64 | 64 eight 8
65 eight 8         | 65 eight 8   | 65 eight 64  | 65 eight 64 | ~~8~~
-0 main 72     72/ | -0 main 72   | -0 main 72   | -0 main 72  | -0 main 72
                   |              |              |             | 65 eight 72
```

(b) How does the terminal output from part (a) above change if the q8 variable is initialized to 8 instead of 88? Only describe what changes.

tid 2 is never waited to join to main...
but since it's last line before main returns, no
output change.

9. (6 POINTS) Apply the Round Robin (RR) scheduling algorithm to the processes described in the table below. Use a timeslice of 3 ms. For each process, calculate the individual turnaround and wait times, as well as the number of times each process is preempted.

For all process arrivals and preemptions, add the given process to the end of the ready queue.

If necessary, break "ties" based on process ID order (i.e., lowest to highest).

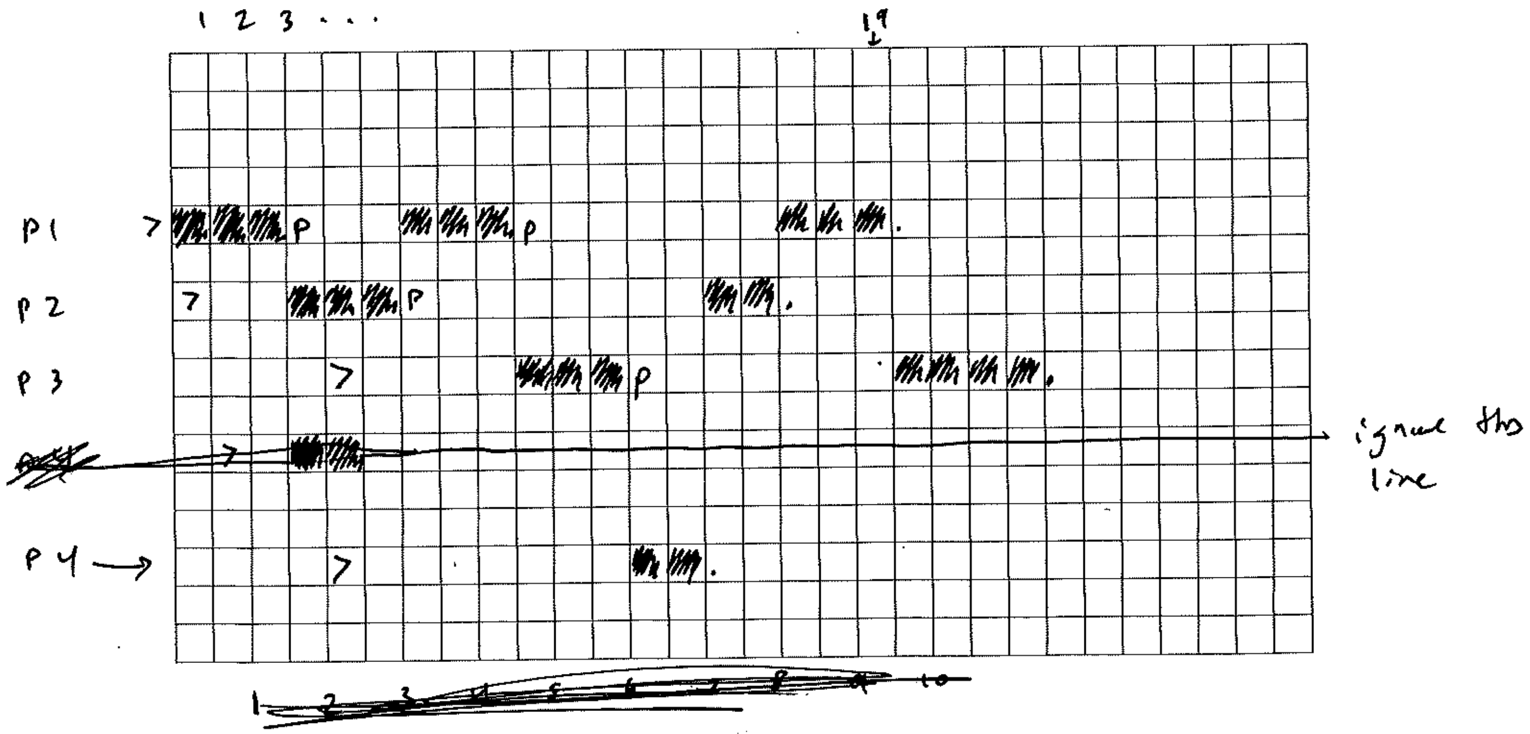
As with the project, if a preemption occurs while there are no processes in the ready queue, do **not** count that as a preemption for that process.

Ignore context switch times and other such overhead.

Fill in the table below with your answers. The grid further below is optional for you to use.

	CPU Burst	Arrival	Turnaround Time	Wait Time	Preemptions
P1	9 ms	0 ms	19	10	2
P2	5 ms	1 ms	16	11	1
P3	7 ms	5 ms	23	16	1
P4	2 ms	5 ms	14	12	0

~~[illegible]~~ ~~[illegible]~~ P3



~~[illegible]~~ 206 + 3510
+ 206
3716

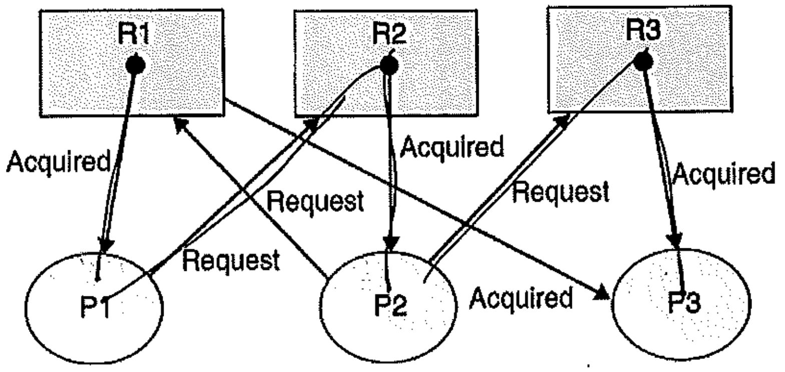
10. **(3 POINTS)** Memory leaks exist in much of the given code on this exam.

    How many bytes are dynamically allocated and not properly deallocated via `free()` throughout this exam? Only include memory directly allocated by the given code, i.e., ignore memory that is dynamically allocated by `printf()`, etc.

    Write the total number of bytes as a number (e.g., 1234). For Question 6, only count memory leaks for the first program execution.

    3510 + 4 + 198 + 440 = $\boxed{3766 \text{ bytes}}$

11. **(1 POINT EXTRA CREDIT)** Is there deadlock in the resource allocation graph shown below? Clearly answer YES or [✓ NO].



12. **(1 POINT EXTRA CREDIT)** When is Professor Goldschmidt's son's birthday? Your answer must be in the format `MMYYYYDD`, e.g., `04202312` if he was born today.

    04202212

USE THE REST OF THIS PAGE FOR ADDITIONAL ANSWERS OR WORK.

USE THIS PAGE FOR ADDITIONAL ANSWERS OR WORK.

DO NOT DETACH THIS PAGE!